A SIMPLE TREE INTERFACE TO X WINDOWS

by

Todd A. Proebsting

Computer Sciences Technical Report #1064

December 1991

A Simple Tree Interface to X Windows

Todd A. Proebsting
todd@cs.wisc.edu

Computer Sciences Department
University of Wisconsin-Madison

December 23, 1991

Abstract

This document describes a very simple abstract interface to display tree data structures in an X window. The interface consists of calls to display a tree, and to delete a tree. In addition, the user must provide an abstract interface to the trees to be displayed. This interface includes a call to determine the maximum number of children a node may have, a call to traverse the tree, and a call to get a string to be displayed to represent the node in the X window.

1 Introduction

To facilitate displaying trees, a very simple interface has been designed to completely hide the details of X windows from the application. Trees may be displayed and destroyed—nothing more, nothing less.

The routines provided automatically initialize the X window interface. It is assumed that the application using these routines does not use any other X window routines—if not, you will have to modify the initialization routines in `Xtree.c`.

Each node in the tree is displayed as a box of text in the X window. The text for a given node is determined by a call to the interface routine, `label()`.

More than one tree may be displayed at a time.

2 Availability

The files, which include an example program, are available via anonymous `ftp` from `kaese.cs.wisc.edu` in the compressed *shar* file `pub/xtree.shar.Z`. Some of the files come from a larger distribution of X routines that are described in the book *The X Window System: Programming and Applications with Xt* by Douglas A. Young (Prentice Hall, Inc.).

The routines require the Xw widgets library developed at Hewlett-Packard. If they are not installed at your site, they are freely available as part of the MIT X distribution from `prep.ai.mit.edu`.

3 A Note on Types (`VoidTree`) and Header Files (`"Xtree.h"`)

The following discussion uses two types: `VoidTree` which refers to the `struct` type of your tree's nodes, and `Xtree` which refers to the type of the handle used by the X tree system.

Your calling routines must include the file `"Xtree.h"` that is in the distribution. Prior to including this file, you must have a `typedef` of the name `VoidTree` to alias the name of your structure's type. In the example Fibonacci routines, that type is called `struct Fib`. The following code is taken from the example that builds and displays trees that represent Fibonacci computations.

```
typedef struct Fib VoidTree;
#include "Xtree.h"

typedef struct Fib {
    int index;
    int v;                /* fib(index) */
    struct Fib *left;     /* Tree representing fib(index-2) */
    struct Fib *right;    /* Tree representing fib(index-1) */
} Fib;
```

4 Routines Provided by the System

4.1 `Xtree newXtree(VoidTree *root, child, label, numchildren, upcall)`

This routine should be called with the root node of a tree to be displayed, and the routines necessary for displaying the tree. The return value is a "handle" to be used when deleting the tree from the window.

The types for `child`, `label`, `numchildren`, and `upcall` are given in §5.

4.2 `void deleteXtree(Xtree treehandle)`

This routine should be called to delete a tree from the X window. The argument is a return value from a previous call to `newXtree()`.

4.3 `void redrawXtree(Xtree treehandle)`

For most *simple* applications, it will not be necessary (or useful) to use `redrawXtree()`. This routine is provided to *force* all of the nodes of a displayed tree to redrawn—in effect, `label()` will be recalled for each displayed node. This will allow updates to a data structure to be displayed without forcing a tree to be completely distroyed and recreated.

5 Routines Provided by the User

The routines provided by the user may be named whatever he likes. They are passed as arguments to `newXtree()`. This allows different types of trees to be simultaneously displayed.

5.1 `VoidTree *child(VoidTree *n, int i)`

This routine must be provided by the user in a call to `newXtree()`. It returns a pointer to the i^{th} child (starting at 0) of the given node. `NULL` pointers are handled just fine, but they are not displayed.

```
static Fib *
childXtree(n, i)
Fib *n;
int i;
{
    assert(n != NULL);
    switch(i) {
    case 0: return n->left;
    case 1: return n->right;
    default: return NULL;
    }
}
```

5.2 `char *label(VoidTree *node, int verbose)`

This routine must be provided by the user in a call to `newXtree()`. It returns a pointer to the string that should label the node in the X window displaying the tree.

The first argument to the call is a pointer to a node in your tree, so you can use it to determine what to print. The second argument refers to whether a short or verbose string should be returned. The value of the second parameter works as a toggle when the given node in the tree is selected with the mouse—the default is for the short label to be used.

Note that embedded new-lines ('\n') will give multi-line nodes in the X tree. (As you would hope.)

The memory management of the strings returned by `label` is handled by the user's program. The memory management of the X tree routines is handled entirely by the X tree routines themselves.

```
static char *
labelXtree(n, verbose)
Fib *n;
int verbose;
{
    static char buf[BUFSIZ];

    assert(n != NULL);
    if (verbose) {
        sprintf(buf, "Fib(%d)\n%d", n->index, n->v);
    } else {
        sprintf(buf, "%d", n->v);
    }
    return buf;
}
```

5.3 int numchildren(VoidTree *node)

This routine must be provided by the user in a call to `newXtree()`. It returns the maximum number of children the given node could have. In the actual node, some may be `NULL`.

```
static int
numchildrenXtree(n)
Fib *n;
{
    assert(n != NULL);
    switch(n->index) {
    case 0:
    case 1:
        return 0;
    default:
        return 2;
    }
}
```

5.4 void upcall(Xtree t, VoidTree *root, VoidTree *node)

It is possible to allow the trees to *react* to the event of being selected with the center button of the mouse. By clicking with the center button of the mouse, a user-supplied routine is called—this routine is given by the `upcall`

argument to the call to `newXtree()`. The arguments provided to the call of `upcall` are the `Xtree` involved, the node selected, and the root of the tree containing the selected node.

These routines are used for *side-effects* such as setting flags (as in the Fibonacci example).

It is legal to pass a `NULL` value to `newXtree()` for the `upcall` argument. If a `NULL` value is passed to `newXtree()`, no action will result from selecting a node with the middle button.

```
static int upcall_flag;

static void
upcallXtree(t, root, f)
Xtree t;
Fib *root;
Fib *f;
{
    upcall_flag = 1;
}

void
main(void)
{
    int i;

    printf("Select tree node with CENTER button to get next tree.\n");
    for (i = 0;; i++) {
        Fib *f;
        Xtree xtree;

        f= fib(i);
        /* create an X tree from a user tree */
        xtree = newXtree(f,
                         childXtree,
                         labelXtree,
                         numchildrenXtree,
                         upcallXtree);

        upcall_flag = 0;
        /* Busy Wait---for user to select tree with center button. */
        while (!upcall_flag) { /* do nothing */ }

        deleteXtree(xtree);     /* delete the given X tree */
    }
}
```

6 User Interface

The window-based user interface is primitive. Three commands are allowed: clicking on a node with one of the three buttons.

Left Button Clicking with the left button on a node of a displayed tree toggles the display of that node between its abbreviated and verbose forms.

Center Button Clicking with the center button on a node of a displayed tree causes the user-supplied routine, `upcall`, to be called with the selected node as one of its arguments. The routine is described in §5.4.

Right Button Clicking with the right button on a node toggles the subtree between being *elided* or not. This helps by allowing uninteresting subtrees to be removed from view. An elided subtree's node will be augmented with "`^^^^^`" above and below the normal text for that node.

7 Multiple, Differing Trees

It is possible to have trees of differing types displayed at the same time. To do this you will need to *overload* the use of `VoidTree`. The best way to do this is use a `typedef void VoidTree`. (If your compiler complains about using `void` then simply use `char`.) Once you have done this, it will be necessary in the routines you supply to cast from this type to the type you are actually expecting. This could have been done for the `numchildrenXtree()` routine in the Fibonacci example as follows:

```
typedef void VoidTree;
#include "xtree.h"

int
numchildrenXtree(VoidTree *node)
{
    Fib *n = (Fib *)node;
    /* the rest of the routine uses n instead of node */
}
```

A Building the Software

Building the software is not complicated. All that is required is that you compile the files `Xtree.c` and `Tree.c`, and link them and the appropriate libraries (`-lXw -lXt -lX11`) to your application.

You should **not** need to change any of the source files to get this to work.

B Running X Remotely

It may be the case that you find yourself running your X application on one machine while sitting at another. Do not despair, with a few simple commands, you can run an X application remotely—you simply tell the remote machine where you are sitting, and you tell the local machine to accept X commands from the remote machine.

The appropriate commands are given below in *italics* (of course, you must use the correct machine names):

On the local machine:
`local-machine%` *xhost +remote-machine-name*
On the remote machine:
`remote-machine%` *setenv DISPLAY local-machine-name:0.0*

After the session is over, you will want to remove the privileges of the remote machine for writing to your machine. This is done as follows:

On the local machine:
`local-machine%` *xhost -remote-machine-name*